Massachusetts Institute of Technology
Lincoln Laboratory

A Model of Network Porosity

James Riordan
Richard P. Lippmann
Sebastian Neumayer
Neal Wagner

Technical Report IA-4

February 4, 2016

This work is sponsored the Air Force Contract FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the authors and are not necessarily endorsed by the United States Government.

Lexington Massachusetts

This page intentionally left blank.

TABLE OF CONTENTS

ABSTRACT

Compartmentalization is one of the standard mechanisms used by defenders to secure enterprise networks. Unfortunately, the compartmentalization process from a security standpoint currently remains more of an art than a science. Even when this art is well executed, the ongoing evolution of the network often violates initial, security-critical design assumptions. Toward improving operational security, MIT Lincoln Laboratory has a collection of metrics which can be used to continuously assess risk within the context of cyber security. One important security metric in this collection is a measure for managing network boundaries and filters or "network porosity." This metric computes the risk inherent to a given network architecture. This technical report describes the model that underlies the network porosity metric and a simulation implementing the model.

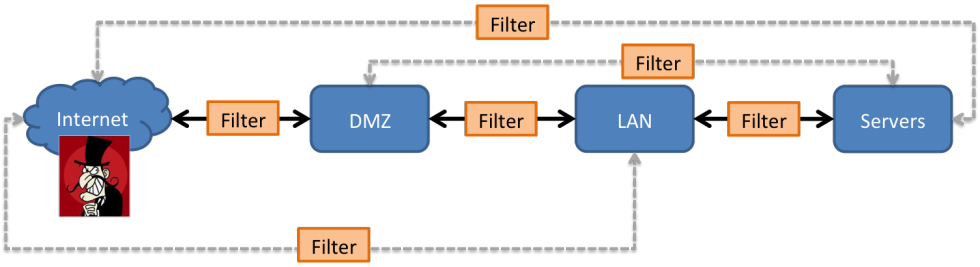

Figure 1. Information Flows

1. INTRODUCTION AND BACKGROUND

The purpose of this technical report is to document this network porosity model and specify a simulation that implements this model. The remainder of this document is organized as follows: Section 2 gives an overview of the network porosity model, Section 3 provides the complete specification of a simulation that implements the porosity model, and Section 4 details event based development techniques that serve to increase the computational efficiency of implemented network porosity simulations (as to make them feasible).

This page intentionally left blank.

2. NETWORK POROSITY MODEL OVERVIEW

We define "network porosity" as the degree to which a network *design* is porous to the flow of an attack over time. This view differs from that of attack graphs [5] which address worst-case attack reachability of a network for a single point in time. Our model addresses the stochastic evolution of services on the network as well as the defensive remediation process, rather than the state of the network at a particular instant in time. The following sections describe the entities and outputs of the network porosity model and conditions that create/aggrevate risk with respect to network porosity.

2.1 NETWORK ARCHITECTURE

We describe a network and its architecture in terms of enclaves, services, functional information flows, and filters.

2.1.1 Enclaves

The first aspect of network architecture concerns the compartmentalization of the network into ***enclaves***. We speak of the attack surface of an enclave as being the union of all the services on devices within the enclave. High-value assets should be placed in *separate* enclaves which are difficult to attack. Enclaves should be constructed so that the collection of services offered within each enclave is as small as possible. We have chosen to model at the level of enclaves, rather than individual devices, for reasons of data availability, performance, and the goal of evaluating the inherent risk in network architectures (as opposed to attack graph analysis of a network at a particular moment in time).

2.1.2 Services

Another aspect of network architecture concerns the use of services that run on enclaves. The term ***service*** is used here in a very general sense as to include programs whose execution can result in attacker control. This includes programs apt to have server-side vulnerabilities such as HTTPD or RPCD and programs apt to have client-side vulnerabilities such as browsers, office suites, PDF viewers, and numerous other common tools. Attackers can propagate either by exploiting vulnerabilities in software or by using trust relationships. The presence, over time, of vulnerabilities within software is taken as an on-going vulnerability-discovery/patch-release process whose properties are informed by history. Service-side and client-side vulnerabilities are distinguished by the rates at which the attacker has the opportunity to exploit them: server-side vulnerabilities may be exploited as the attacker desires and so attacks are essentially instantaneous whereas client-side vulnerabilities require the action of a network user and thus depend upon the estimated aggregate actions of network users.

2.1.3 Functional Information Flows

Functional information flows are an aspect of network architecture that specifies communication pathways between enclaves. We use the term "functional information flow" to focus

on information flows where the source is the enclave from which the attack originated and the destination is the enclave that was compromised. Network and functional information flows are depicted in Figure 1 although a more formal description of functional information flows is given in Section 3.2.1 below.

In the case of server-side vulnerabilities, functional information flows often correspond to the network reachability of a service. In the case of client-side vulnerabilities, functional and physical information flows may be different. Consider an attack scenario on the network based on Figure 1 in which an attacker emails a malicious PDF document which is read using a vulnerable PDF reader by a user on the LAN. The result is an attack which starts on the Internet and compromises the LAN (this step compromises *only* the LAN). We describe this as a functional information flow between the Internet and the LAN; allowing for other typical information flows we end up with the information flow graph depicted in Figure 1. We contrast this with the physical information flow wherein the malicious message deposited on a mail server on the DMZ, pulled in to a mail server situated on the Server enclave, and finally pulled from the mail server back to the LAN. In this scenario, neither the DMZ nor the Server enclave have been compromised. As such, reachability of services apt to expose client-side vulnerabilities in a given enclave is expressed in terms of other enclaves that can functionally deliver, or have delivered, content that could exploit the vulnerabilities.

In practice, nearly all functional information flows that concern client-side vulnerabilities will stem from the delivery of email, access to the Internet, or access to an internal document or data repository. If any of these are possible from a given enclave, then there exists an information flow from the Internet or repository to the enclave. We note explicitly that the notion of functional information flows applies to information transfer between air-gapped systems.

Trust relationships between devices (e.g. a trust relationship created by a domain controller or ssh) can also be considered as an information flow which allow an attacker to propagate from one enclave to another. In practice, trust relationships will be implemented as services that are never patched.

This architecture is depicted with the black lines in Figure 1. We generally express such networks in terms of the services running in each enclave as well as the routing and firewall rules between the enclaves.

This network diagram does not naturally express the full range of paths that an attack might traverse as it does not express transitivity of routing (e.g. the server LAN might be able to contact the Internet despite the fact that there is not a direct line between them) or complex information flows (mail delivered from the Internet to the DMZ, pulled into the server and read from the LAN). Analysis of such networks and creation of attack graphs then is addressed in [4]. We will thus base the metric on functional information flows as depicted through union of the black and dashed gray lines in Figure 1.

2.1.4 Filters

The final aspect of network architecture concerns ***filtering mechanisms*** used to mitigate risk inherent to the permitted functional information flow. These filtering mechanisms fall into two categories. The first is filtering based on particular subsets of enclaves that are permitted to

communicate with each other. This includes both Internet black-listing (destinations as presumed malicious) and Internet white-listing (destinations presumed benign). We would note that for the purposes of evaluating network architectures, we need to limit the scope of source/destination based filtering lest we turn the model into a full probabilistic attack graph analysis of the network whose data requirements are currently unrealistic. The second is filtering based on content. This addresses such filters as generic application-layer filtering, spam and phishing filters, general content filtering, intrusion prevention systems, and data leakage protection systems.

2.2 ATTACKER

The model specifies an attacker who gains access to internal enclaves by exploiting vulnerable services and then attempts to pivot to other network enclaves via allowed information flows (e.g. open ports/protocols or manually transported media). When a vulnerability is discovered in a service, the attacker begins to develop an exploit. Once the exploit is developed, the attacker will use the exploit to penetrate a network and then spread throughout the network at each opportunity. The movement of an attacker from one (compromised) enclave to another (uncompromised) enclave is is depicted in Figure 2. In the figure, the attacker has compromised enclave_1 and attempts to spread to enclave_2. For this example, the conditions that allow an attacker to spread are as follows:

- there is a service running on enclave_2 that has a vulnerability,
- this vulnerable service onenclave_2 can be reached from enclave_1 (e.g. there is a functional information flow from enclave_1 to enclave_2 via the service),
- the attacker has an exploit for the vulnerability,
- the exploit is not prevented by the filter, and
- there is an opportunity for the attacker to attack.

The above attacker model is based on commonly observed network attacks. The following provides a detailed description of these attacks:

- External attackers continuously probe networks for servers with open ports and then attack and compromise these servers. There may be many reasons why a server can be compromised. Known vulnerabilities may not be immediately patched, servers may be misconfigured, attackers may have discovered a previously unknown zero-day vulnerability, there may be no mitigation for a vulnerability, or it may not be possible to apply a mitigation. Restricting information flows within a network provides a second layer of defense to other controls including using anti-virus software on hosts, patching vulnerabilities and ensuring that software is configured properly.
- After attackers compromise a server, they can compromise other devices in the same subnet or protected enclave. They probe attached firewalls and routers for open ports and attempt to compromise additional servers with open ports in other enclaves or other devices that can

be reached through information flows. This allows attackers to pivot and penetrate deeper into internal enclaves. This process of attacking servers in other enclaves can continue until attackers have reached all enclaves. Limiting accessible services between enclaves provides a second layer of defense that complements other controls.

- Additional filtration mechanisms may be utilized to help stem the propagation of attack between enclaves where information flows of a particular type (service) are permitted:
  - An attacker originating from a known malicious site connects to a vulnerable device. A blacklist or whitelist would prevent this attacker from connecting to this device. Blacklists and whitelists provide another layer of defense to other controls including using antivirus software on hosts, patching vulnerabilities and making sure software is configured properly.
  - A network device connects to a known malicious site that enables a client-side attack. A blacklist or whitelist would prevent this device from connecting to the known malicious site. Blacklists and whitelists provide another layer of defense to other controls.
  - A network device receives known malicious content via email that enables a client-side attack. A spam and malware filter would prevent this content from reaching its destination. Content filtering provides another layer of defense to other controls.
  - Attackers perform server-side attacks by sending malicious content to exposed applications and devices. An Intrusion Prevention System (IPS) may block the exploit by filtering malicious content.
  - Attackers can travel to and from an air gapped network via a Personal Electronic Device (PED) such as a USB stick. Removing the ability to use PEDs on air gapped enclaves, e.g., by removing USB ports, can be modeled as a filter on this information flow.
  - Untrustworthy persons may intentionally try to exfiltrate known sensitive data to external networks. People may also unintentionally leak known sensitive data. Content filters that check for and remove outgoing sensitive data would prevent this data leakage. Content filtering provides another layer of defense to other controls including managing network access.

2.3 MODEL OUTPUT

Ultimately risk reduction is the goal of the metrics efforts. Risk concerns the impact of events to an organization weighted by the probabilities that those events will occur. General purpose impact modeling is outside the scope of this paper. As such, we focus on event probabilities. The output of the network porosity model is a stream of timestamped events describing enclave compromise and enclave cleaning. Enclave compromise events describe the source (enclave) and vector (service) of the compromise. This stream can be used to compute a variety of outputs including but not necessarily limited to the following:

- percentage compromise times for each enclave,

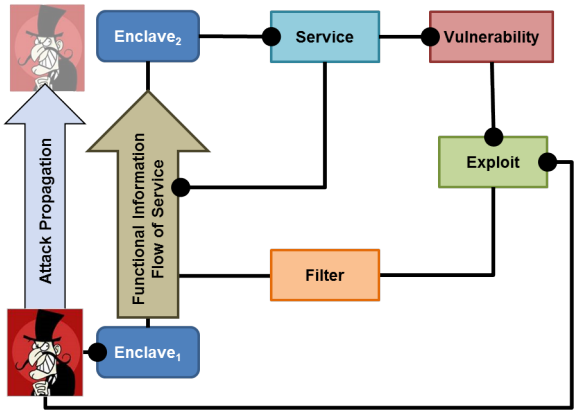

Figure 2. Attack Propagation

- the cumulative density functions (CDF) of compromise times for each enclave,
- the distribution of percent time in each element of the power set of enclaves (where elements of the power set describe the set of enclaves compromised at a given time),
- a Markov process approximating porosity on the power set of enclaves (again where elements of the power set describe the set of enclaves compromised at a given time),
- mean/average time until enclaves are compromised by an attacker starting with on the Internet.

2.4 RISK CONDITIONS

The following are examples of common events that create or aggravate risk conditions with respect to network porosity:

- Instead of putting all web servers open to the Internet into a DMZ network, they are placed in a large common internal network shared by all users. If any server is compromised, this could allow attackers to compromise other systems on the internal network.
- Instead of filtering communications between servers in the DMZ and the internal network using middle-ware application proxies, direct connections from the Internet are allowed to database, email, and FTP servers on computers in the internal network. If DMZ servers are compromised, these direct connections could allow attackers to compromise systems on the internal network.
- An administrator installs a database on the internal network and allows direct incoming connections from the Internet to facilitate remote testing. These open ports remain open. This opens up a direct attack path into the internal network.

- In a small, but growing high technology company, all systems are on one large network including payroll, sales, administration, and engineering. This allows an attacker who penetrates into the internal network to access valuable intellectual property in engineering, to steal proprietary plans from the administrators, and possibly to arrange illicit monetary transfers from the payroll or sales departments.
- A person uses a USB stick to move files to and from an air gapped network; this information flow can result in the compromise of the air gapped enclave or other internal enclave.
- Although different high-value assets are separated into different enclaves, many information flows could still exist between these enclaves. This could allow attackers to easily pivot between subnets.
- An attack originating from a site known to conduct SSH password guessing targets an SSH server running on a network device, initiates a connection, correctly guesses a password, and compromises the device; a blacklist would have blocked this brute force attack.
- An attacker targets an SSH server that can be defaulted to use an older protocol version that can be exploited; a properly implemented content filter would have blocked this protocol roll-back attack. In principle this attack could be blocked by using software that is not vulnerable to such roll-back attacks.
- A person browses to a gambling site containing malware; a blacklist would have prevented this person from visiting this site, which is not related to mission function.
- An untrustworthy person exfiltrates known sensitive data by uploading it to an external FTP site; a properly implemented content filter would have prevented this data leakage.
- An untrained person attaches a sensitive document to an email sent to an outside organization; a properly implemented content filter would have prevented this data leakage.
- A content filter has been deployed, but in such a fashion that it can be avoided by an attacker.
- Once an exploit runs on a victim computer, it attempts to download larger components of an attacker toolkit from a known malicious site; a blacklist would block this download and alert the network administrators.
- A person receives a phishing email containing a malicious attachment or a link to a malicious site; a properly implemented spam filter would have blocked this e-mail.
- An attacker compromises a domain controller and uses the gained trust relationships to compromise additional machines on the LAN.

All the above events may lead to a device in an enclave becoming compromised. The attacker may then continue to spread and compromise other reachable machines on other enclaves on the network.

3. OVERVIEW, DESIGN, AND DETAILS (ODD) FOR SIMULATION OF NETWORK POROSITY MODEL

This section will provide details on the components, procedures, data requirements, and parameters required to instantiate the network porosity model. These required parameters and observations are listed in Tables 1 and 2 that characterize defender and attacker capabilities.

We assume both the arrivals of vulnerabilities for a particular service and the disentrenching of an attacker from a particular enclave via cleaning follow Poisson processes as described in [7, pages 30-31]. These parameters are used in a simulation to compute the risk metric.

Default values for the Δ parameters in Table 1 can be explicitly specified for each service or enclave. Specifying only these default values makes it possible to compute the risk metric without providing different interarrival times for each service or enclave.

The Overview, Design, and Details (ODD) is a widely-used protocol for specification of simulation models of complex systems [1]. Although the ODD protocol was originally intended for individual-based or agent-based models (ABM), we adopt this protocol for simulations which need not necessarily belong to the ABM paradigm. The main objective of the ODD protocol is to provide a template for thorough model description that would allow for reproducibility of simulation-based experiments. The purpose of this document is then to provide such a description, utilizing the ODD protocol, of a simulation model that can be used to compute the operational risk metric of BOUND-N.

The ODD protocol template consists of the following seven parts:

1. **Purpose:** consists of a summary of the problem being solved and the overall objectives of the model. We present this summary in Section 3.1.
2. **Entities, State Variables, and Scales**: refers to model state specification to such a degree of detail, that were the simulation to be stopped with its current state saved, no information would be lost and the simulation could be restarted in exactly the same state some time later. We discuss these details in Section 3.2. We also provide a summary in Table 3.
3. **Process Overview and Scheduling**: is a description of all processes and their order of execution in terms of schedules, specified with pseudo code. These details are discussed in Section 3.3.
4. **Design Concepts**: is a list of eleven model design points. These points are discussed in detail in Section 3.4.
5. **Initialization**: refers to the specification of all the initial conditions (the state at the start of every simulation run). We discuss initialization in Section 3.5.
6. **Input Data**: refers to the specification of any inputs to the model from external sources. Discussion of these inputs we provide in Section 3.6.

7. **Sub-models:** is a detailed description of the sub-models that represent the processes listed in the Process Overview and Scheduling above. We discuss possible sub-models in Section 3.7.

In addition to describing the simulation model via the ODD protocol, we also discuss implementation techniques that address model efficiency and verification/replication.

3.1 SIMULATION MODEL PURPOSE

The simulation is intended to model the security risk to a computer network from outside attackers who gain a foothold on an internal enclave of the network and then try to pivot to other network enclaves by compromising exposed devices through ports/protocols or compositions of multiple ports/protocols (i.e., that may correspond to functional information flows) allowed between enclaves. Recall that an enclave E refers to a collection of networked devices with an associated network boundary B when the following two properties are satisfied:

- any information flow of any device outside of E must traverse B to reach any device inside E,
- any pair of devices inside E can reach each other (i.e., there exists at least one protocol that they can use to exchange information) and any possible information flow between them does not have to traverse B.

Recall that a network boundary we have defined as any composition of filters, firewalls, gateways, and any other mechanisms used to monitor and regulate information flows between a pair of devices. See Figure 3 for an example of two enclaves separated by their respective network boundaries. The simulation is provided for computation of the enclave compromise and clean events for a particular network with a given network architecture. A network architecture includes the set of enclaves on the network, the set of services running on these enclaves, information flows between these enclaves, filters that serve to control information flows between enclaves, and the set of resources that can be accessed via these enclaves. The simulation is intended to compute the expected value of resource loss for a network with a given architecture over a given length of time.

3.2 ENTITIES, STATE VARIABLES, AND SCALES

The following is a description of entities to be modeled within the simulation, the state variables/attributes, and the temporal and spatial scales that characterize these entities.

3.2.1 Network environment

A network environment is modeled in which there exist several connected enclaves such that communications can flow between enclaves based on a given communications topology (i.e., set of communication pathways between enclaves). A network environment is characterized by the following state variables.

- Communications topology: a set of enclaves and communication pathways that connect these enclaves.

Here, communication pathways between enclaves do not represent physical connections between enclaves, but rather represent *functional information flows*. A functional information flow refers to any communication pathway between two enclaves and includes pathways in which the source and destination are directly connected (by a physical line), transitive communication pathways (e.g. a server enclave being able to communicate with the Internet despite the fact that no direct line between these two enclaves exist), and more complex information flows (e.g. an email is sent from the Internet enclave that arrives at a DMZ enclave, is pulled by an email server to a server enclave, and is finally downloaded and read by a user account in the LAN enclave). For complex information flows, such as the email information flow just described, it is important to note that an attack coming from a particular source enclave (such as the Internet) may eventually compromise the destination enclave (in this example the LAN enclave) *without compromising any of the intermediate enclaves between* these two enclaves (in this example the intermediate enclaves are the DMZ and server enclaves).

For the purposes of the simulation, a functional information flow is modeled only as a communication pathway from the source enclave to the destination enclave. Intermediate enclaves between the source and destination are not modeled.

3.2.2 Enclave

A network enclave is modeled. As discussed in section 3.1, an enclave is defined as a group of network devices with homogeneous reachability. An enclave is connected to other enclaves via functional information flows as described above. A connection from a source enclave to a destination enclave is made through one or more services running on the destination enclave. An enclave has access to one or more resources that have value and, thus, are subject to cyber attacks that seek to compromise the enclave and extract some value from these resources. For this model, we assume that an enclave e becomes compromised when a service running on e has had an exploit event arrive on the service and a functional information flow connects an already compromised enclave e_{comp} to e via this service. We assume that there exists a process to cleanse an enclave, i.e., disentrench an ensconced attacker from the enclave via malware tools, re-imaging, or other similar measures.

An enclave is characterized by the following state variables.

- The set of services running on the enclave.
- The set of resources that can be accessed by the enclave.
- A flag that signals whether or not the enclave is compromised.
- The average completion time for cleansing the enclave, Δ_{clean}. The completion of enclave cleansing is modeled as a Poisson process and Δ_{clean} represents an average inter-arrival time for completion of cleansing of the enclave. We assume that the cleansing process for each enclave is independent and that different enclaves may have different values specified for Δ_{clean}.

It is important to note that cleansing completion events for an enclave are modeled as arriving at any time, whether or not the enclave is actually compromised. It is not necessary to model cleansing events as arriving only after enclave compromises. This is because upon a compromise, an attacker may become disentrenched by various network hygiene practices, such as software updates/patches or periodic re-imaging, as well as by explicit cleansing.

An enclave can be either compromised or uncompromised. An uncompromised enclave becomes compromised when an exploit event arrives on a service on the uncompromised enclave and an information flow connects a compromised enclave to a service in the uncompromised enclave.

For each enclave e we have an enumeration of services $S(e)$ where each service $s \in S$ behaves according to the above description. An attacker compromises an uncompromised enclave when an exploit event arrives on a service running on that enclave and an information flow connects a compromised enclave to that service in the uncompromised enclave. We note a process may exist on multiple enclaves, but the vulnerability process is identical for all processes of the same type.

A compromised enclave becomes uncompromised when the defender disentrenches the attacker from the enclave. This can happen via malware removal tools, reimaging, etc. We assume a process to disentrench an attacker from enclave e occurs on each enclave according to a Poisson process with average interarrival time $\Delta_{\text{clean}}(e)$. We assume the disentrenchment process for each enclave is independent.

3.2.3 Service

In this section we describe the model that determines the times when an enclave running a particular service, s, can be compromised by an attacker who exploits a vulnerability in that service. We begin with a high level description of a service. Over time, multiple versions of a service will be developed. For each service version, we assume vulnerabilities will be discovered over time. When a service is upgraded to the next version, we assume the vulnerabilities in the previous version no longer exist in the current version. For each vulnerability we assume there exists exactly one possible exploit. See Figure 4 for a simple depiction of a service process over time.

We now further describe a service by describing how the arrival of patches, vulnerabilities, and exploits occur over time. A service may have multiple vulnerabilities. We assume that each vulnerability may correspond to exactly one exploit that may be used to compromise the service (and thus compromise the enclave that the service is running on). We also assume that there exists a process to update/patch a service. We assume each service initially has no vulnerability present. New vulnerabilities for service s arrive (i.e. become known) according to a Poisson process with average inter-arrival time $\Delta_{\text{vuln}}(s)$. For every vulnerability, the time to develop an exploit once the vulnerability has arrived is exponentially distributed with mean $\Delta_{\text{dev}}(s)$. It is important to note that whenever a vulnerability arrives on a service, an abstract "clock" governing the time for an exploit to be developed for this vulnerability is started immediately.

When an exploit is developed for any vulnerability, we assume the service can be exploited at times corresponding to a Poisson process with average inter-arrival time $\Delta_{\text{exploit}}(s)$. A small $\Delta_{\text{exploit}}(s)$ implies once an exploit is developed, the exploit can be used shortly thereafter (e.g. as

in a server-side exploit). A large $\Delta_{\text{exploit}}(s)$ implies opportunities to use exploits are infrequent. These exploits might represent some type of client-side exploit where the attacker must wait for a user action that initiates the exploit and, thus, may be forced to wait a longer time before the exploit is executed.

We also assume the service is patched to a new version according to a Poisson process with average interarrival time $\Delta_{\text{patch}}(s)$. We assume when the service is patched to a new version, all vulnerabilities are removed regardless of whether an exploit has been developed for those vulnerabilities or not. Hence, when patching occurs, the service is considered to have no vulnerabilities regardless of its current state. We also note that this patching is assumed to be synchronized across all instances of services s running on enclaves. That is, every service s is patched at the same time regardless of which enclave it belongs to.

In [summary, a service is characterized by the following state variables. For each of the below variables that specify an average inter-arrival time for events of a particular type, event arrival is modeled according to a Poisson process unless otherwise stated.

- The average inter-arrival time for new vulnerabilities on the service to become known, Δ_{vuln}.
- The average inter-arrival time for an exploit to be developed for a currently present (known) vulnerability on the service, Δ_{dev}.
- The average inter-arrival time for the service to be exploited once an exploit has been developed for a vulnerability present on the service, Δ_{exploit}.
- The average inter-arrival time for a service with one or more known vulnerabilities to be patched/updated, Δ_{patch}.

We assume that the processes governing vulnerability, exploit development, exploit, and patch arrival are independent for each service and that different services may have different values specified for their respective inter-arrival time variables above.

3.2.4 Setting parameters for Each Service

We now describe how parameters for each service can be informed by historical data. We first describe how $\Delta_{\text{patch}}(s)$ can be set for a particular service s and then describe how $\Delta_{\text{vuln}}(s)$ can be set. The basic idea of how to inform both these parameters is to look at a sufficiently long historical window for the service s and take the (possibly weighted) average time between relevant events. This average time between events can then be used as a parameter for the service. More specifically, consider $\Delta_{\text{patch}}(s)$. For a long historical window of length L, we gather the number of patches released for service s, denoted by $\#$ *of patches*, as well as the total length of time the service had at least one unpatched known vulnerability, denoted by K. We can then simply assign the average patch interarrival time to be $\Delta_{\text{patch}}(s) = \frac{K}{\#\ of\ patches}$. For the case of setting $\Delta_{\text{vuln}}(s)$ we perform a slightly more complicated calculation that weights each vulnerability in the historical window by some function of its CVSS score. More specifically, consider the set of all known vulnerabilities for service s over a sufficiently long historical period of length L. Let this set

of vulnerabilities be denoted by $V(s, L)$. To account for the fact that different vulnerabilities have different severities we will use the normalization found in Exploitation of Known Vulnerabilities [6] and weight each $v \in V(s, L)$ as a function of its CVSS score. We simply set $\Delta_{\text{vuln}}(s)$ to be total length of the historical window divided by this weighted sum.

$$\Delta_{\text{vuln}}(s) = \frac{L}{\sum_{v \in V(s,L)} \left(\frac{CVSS(v)}{10}\right)^2}$$

3.2.5 Filter

A filter acting on a service is modeled. A filter refers to a defensive technology that attempts to block malicious communications on a service. Examples include firewalls or filters that employ communication blacklisting (disallowing communications from known malicious sources) or whitelisting (allowing only communications from known benign sources) and filters that block communications based on their potential for containing malicious content, such as spam and phishing filters, application filters, and intrusion detection systems. See Figure 5 for a depiction of how filters are applied to a service.

A filter acting on a service serves to potentially block exploits, that is there is some probability a filter blocks a particular exploit. A filter is characterized by the following state variables.

- The ineffectiveness of the filter for service s, $FI(type, s)$, which specifies a probability that an exploit event on s is not blocked by the filter *type*. The parameter *type* specifies the filter type.

The filter ineffectiveness variable employs the convention that larger values are worse from a security perspective. For example a service with no filtering specifies a filter ineffectiveness of 1 while a filter that perfectly blocks all exploit events on the service specifies a value of 0. For example, if the protocol is simply not routed between the enclaves then this conditional probability is 0 and if the protocol is routed between the enclaves without any filtration then the conditional probability is 1. It is also important to note that multiple filters of the same type applied to the same service in serial are effectively no different than a single filter of that type for that service. This is because filters of the same type allow the same sets of exploits through and thus, applying them multiple times will not serve to block additional exploits.

3.2.6 Resource valuation

A resource is modeled. A resource is an abstract item that is accessible from one or more enclaves. A resource may be data, such as electronic files, an application or functional service, such as an application that allows authorized users to execute an organization-specific action (e.g. submit a purchase order), or any other item of value that an enclave can access. A resource has some value that represents its relative importance to the organization and its mission.

We assume that opportunities for theft of a resource arise and, when they do, an attacker who has compromised one or more enclaves that allow access to the resource will extract or pilfer

some amount of value from the resource at each opportunity. After multiple theft opportunities and subsequent resource value extractions, the full value of the resource will be lost. As described in section 3.1, the purpose of the simulation is to compute, the expected total resource loss for a network with a given network architecture.

A resource and its theft are characterized by the following state variables.

- The total value of the resource, v_0. This value represents its relative importance to the organization at the beginning of the simulated time period and assumes that no theft on the resource has occurred.
- The average inter-arrival time for theft opportunities on the resource, $\Delta_{\text{opportunity}}$ (recall that a resource can be stolen only when an opportunity to steal it arrives). We assume that theft opportunities arrive on a resource according to a Poisson process and that resource theft opportunities for different resources are independent and may specify different values for $\Delta_{\text{opportunity}}$.
- The amount of value extracted at each theft of the resource, x. This is specified at a value in $[0, v_0]$. For resources that may be slowly pilfered over time, such as an electronic debit account with a maximum purchase/withdrawal limit, this may be specified as some fraction of v_0. For resources that need only be stolen once, such as intellectual property, this may be specified as v_0.

We assume that when attackers have compromised an enclave, they can access any resource that the enclave can access. If the attackers have compromised multiple enclaves simultaneously, then they have access to the union of resources that the compromised enclaves can access. It is important to note that compromising multiple enclaves simultaneously with access to the same resource offers no advantage to the attacker, i.e., an attacker can only steal value from the resource once per theft opportunity. The total loss to a resource over a simulated time period is computed as the number of thefts of the resource multiplied by its extractable value, x, but not exceeding its total value, v_0.

3.2.7 Temporal and spatial scales

For the simulation, the spatial scale of the various entities does not matter. The temporal scale matters only in that simulation time units be fine-grained enough to capture event arrivals that follow a Poisson process. For many network scenarios, a time unit of milliseconds (or larger) is usually sufficient. It is important to note that an event-based simulation may be more efficient from a computational perspective than a time-based simulation since network events may be relatively sparse with respect to the specified simulation time unit. For example if network events occur hourly and the time unit is milliseconds, there may be many consecutive time units for which no event occurs thus requiring unnecessary computation.

Additionally, some network events do not cause the state of the network to change. For example, an exploit event arriving on a service that runs only on an already compromised enclave does not cause the network state to change and, thus this event may be disregarded. In general, only events that change the state of the network, e.g., by causing an uncompromised enclave to

become compromised or by causing a vulnerable service to have an exploit developed for it, should be considered.

3.3 PROCESS OVERVIEW AND SCHEDULING

The simulation is intended to compute the expected value of resource loss for a network with a given architecture over a given window of time. We will heretofore refer to the window of time in which to compute resource loss as the measurement window W. A simulation run is executed as follows.

1. Instantiate all model entities described above with the exception of resources entities. This includes the specification of all state variables for each entity. Before simulation execution starts, all enclaves are initially uncompromised and services running on enclaves have no vulnerabilities present.

2. Start the simulation execution, that is start the applicable Poisson processes to generate events (e.g. arrivals of vulnerabilities, exploit development, exploits, patches) for the entities on the network. Note that because resources have not yet been instantiated, no events for resources (e.g. theft opportunity events) are generated.

3. It is not always possible to wait until the simulation execution reaches a "steady state" with respect to being in a compromised state, i.e., when the fraction of time that enclave is compromised remains unchanged. To gain some intuition, consider the strategy for determining if an enclave has reached steady state based on how the following quantity changes over consecutive runs:

$$\boxed{\text{Fraction}_{\text{comp}}(E) = \frac{t_{\text{comp}}(E)}{T}} \tag{1}$$

   where $t_{\text{comp}}(E)$ represents the total time that enclave E has been compromised, T is the total simulation time, and $Fraction_{\text{comp}}(E)$ represents the fraction of time E has been compromised. Suppose we say that the steady state is reached when the difference between $Fraction_{\text{comp}}(E)$ over any two consecutive runs of the simulation is a non-increasing number and is at most δ, for some $0 < \delta < 1$. It is not clear, however, that it is not possible to construct a system of enclaves such that, for any $0 < \delta < 1$, $\text{Fraction}_{\text{comp}}(E)$ is close to 0 over any finite sequence of runs required to satisfy this steady-state requirement, but in actuality $\text{Fraction}_{\text{comp}}(E) = 1$ at steady state. Therefore, it is best to specify the exact number of runs to be made, e.g., 1,000,000.

4. Once all enclaves have reached steady state as described above, instantiate resources in the simulation environment and start the applicable Poisson processes to generate events for resources and their theft. It is at this point in the simulation run that the measurement window W begins. It is important to note that resources and their theft are not modeled until the network has reached steady state in order to capture the fact that the attacker may be already entrenched on an internal enclave of the network as well as on an outer enclave that the organizational network communicates with. This is a critical aspect of the model

since the reachability of internal enclaves to each other is important, not just reachability from an outer enclave (e.g. the Internet) to the internal enclaves. Once in steady state, the resources appear and the attacker attempts to steal network resources as opportunities arise.

Monte Carlo simulation is utilized to compute the expected resource loss for a given network architecture over the measurement window by repeating the above steps to execute multiple simulation runs. Expected resource loss is calculated by averaging the resource loss over all simulation trials. If computationally feasible, the number of trials executed should be on the order of 10^4 or greater to ensure a reliable characterization of expected resource loss.

3.4 DESIGN CONCEPTS

3.4.1 Basic principles

As described in Section 3.2.3, a continuous-time Markov chain is employed to model how the state of a service running on an enclave changes over time. This is used to capture relevant state transitions such as vulnerabilities being discovered for the service, exploits being developed and then arriving to exploit the service, and patches being applied to the service. This model of a service together with the interactions of services with the other simulation entities are intended to compute the operational risk for a network with respect to the management of network boundaries and filters used to partition a network into enclaves.

3.4.2 Emergence

The primary result that emerges from execution of the simulation model is the expected resource loss for a network with a given network architecture over a given length/window of time. A secondary result that may also be of interest is the fraction of time that an enclave is compromised over a given length/window of time, which may or may not be indicative of what happens to the system in the steady state (Section 3.3, Equation 1).

3.4.3 Adaptation

For this model, agents/entities do not exhibit adaptive behavior.

3.4.4 Objectives

Because agents in this model do not exhibit adaptive behavior, their objectives are not modeled.

3.4.5 Learning

Learning is not relevant for this model.

3.4.6 Prediction

Prediction is not relevant for this model.

3.4.7 Sensing

Sensing is not relevant for this model.

3.4.8 Interaction

Several interactions between simulation entities occur. Filters directly affect services by altering the rate at which exploit events arrive on these services. Services directly affect enclaves by causing them to be more or less vulnerable to exploits. Enclaves and the network environment (i.e. the communication topology of the network) directly affect resources by making them more or less likely to be stolen over a given window of time.

3.4.9 Stochasticity

Several simulation entities are modeled probabilistically in which events affecting entity state arrive according to a Poisson process with a specified average inter-arrival time.

3.4.10 Collectives

An enclave is specified in part by the set (collective) of services that it runs and the set of resources it has access to. A service is specified in part by the set of filters that act on it. These collectives are defined by the modeler during the initialization of a simulation run.

3.4.11 Observation

Data collected from a single simulation run includes the following.

- The resource loss to a network with a given network architecture over a given window of time (Eq. **??**).
- For each enclave, the fraction of time the enclave is compromised once the simulation has reached steady state (Eq. 1).

These observations are collected and averaged over multiple runs ($\approx 10^4$) to compute their expected values.

3.5 INITIALIZATION

A simulation run is initialized by instantiating a network architecture without resource entities, executing the simulation until all enclaves have reached steady state, and then instantiating the resource entities within the architecture (as described in Section 3.3).

3.6 INPUT DATA

While the model does not explicitly require input data from network logs, audits, and/or monitoring systems, simulation results will more accurately capture the operational risk to a real

network being modeled if the specification of relevant simulation entity state variables (e.g. Δ_{clean}, Δ_{vuln}, Δ_{patch} described in Section 3.1) is informed by such data.

3.7 SUB-MODELS

The simulation does not require any sub-models to characterize processes or sub-processes given in Section 3.3. However, the specification of entity state variables may be informed by network observations (as described in Section 3.6) or by sub-models that seek to characterize lower level details relevant to a particular state variable. For example, in order to specify the average inter-arrival of network cleansing events, Δ_{clean} (Section 3.2.2), a sub-model that models the various details and mechanisms by which an attacker may be disentrenched from an enclave can be built. Monte Carlo experiments can be executed on this sub-model and results aggregated to specify Δ_{clean} in this model.

Required Parameters	
Notation	**Description**
W_t	*Defender Target Duration* - the maximum delay for observing any change in the network architecture. For this metric, defenders must observe the network to determine its current architecture including enclaves and filtering between enclaves and to outside sites. We assume that defenders must observe the network architecture fast enough to detect any architectural change within a duration W_t after it occurs. W_t must be short compared to both the interarrival time for attacks and the interarrival time for changes in network architecture to guarantee that risk is estimated accurately.
$FI\,(type,\ s)$	*Filter Ineffectiveness* - the assumed fraction of exploits blocked for the service s for a particular filter *type*. A perfect filter has *Filter Ineffectiveness* equal to 0 whereas a useless filter has *Filter Ineffectiveness* equal to 1 (retaining the convention that big is bad). In general the ineffectiveness of the filter is largely a function of s. This parameter focuses on a collection of technologies including "reverse" or "inbound" proxies, black and white listing, intrusion prevention systems, data leakage prevention systems, and email malware filters.
$\Delta_{\text{vuln}}(s)$	*Interarrival Time Of New Vulnerabilities* - assumed interarrival time between new vulnerabilities on service s. The interarrival time is exponentially distrubuted with mean $\Delta_{\text{vuln}}(s)$.
$\Delta_{\text{dev}}(s)$	*Average time to develop an exploit for a particular vulnerability* - assumed time to develop an exploit for a particular vulnerability on service s. The time is exponentially distributed with mean $\Delta_{\text{dev}}(s)$.
$\Delta_{\text{exploit}}(s)$	*Interarrival Time Between Exploitation* - default assumed interarrival time for the execution of exploits on service s. The interarrival time is exponentially distributed with mean $\Delta_{\text{exploit}}(s)$. The interarrival time for execution of server side exploits is typically much smaller than the interarrival time for execution of client side exploits.
$\Delta_{\text{exploit}}(s, e_{\text{s}}, e_{\text{d}})$	*Interarrival Time Between Exploitation* - assumed interarrival time for the execution of exploits on service s from source enclave e_{s} to destination enclave e_{d}. This is a specialization of the previous parameter to allow for
$\Delta_{\text{patch}}(s)$	*Interarrival Time To Patch Service* - assumed interarrival time for patching service s. The interarrival time is exponentially distributed with mean $\Delta_{\text{patch}}(s)$. Immediately after patching, the service is assumed to have 0 vulnerabilities.
$\Delta_{\text{clean}}(e)$	*Interarrival Times Between Enclave Cleaning* - assumed interarrival time between the disentrenchment of the attacker from enclave e. The interarrival time is exponentially distributed with mean $\Delta_{\text{clean}}(e)$. We assume the disentrenchment/reset process for each enclave is independent.

TABLE 1

Required and Optional Parameters for Maturity Metrics

Required Observation Types	
Notation	**Description**
E	The set of enclaves in the organization. Enclaves are determined by examining filter and firewall rules (excluding host-based control mechanisms). It is often devices in a subnet protected by firewalls, gateways, and proxy filters.
S	The software inventory should be used to determine the set of all services running on the network. A service can be on a single device or span multiple devices.
$Map_{\text{DevicesToEnclaves}}$	For each device in the inventory of devices found in a hardware inventory, determine what enclave it belongs to
services(e)	Set of services running in enclave e
flow(e_i, e_j, s)	Determine is there exists an information flow between enclave e_i and service s in enclave e_j
filters(e_i, e_j, s)	Determine what filter types are used on the information flow between enclave e_i and service s in enclave e_j

TABLE 2

Required Observation Types for Maturity Metrics

Entities, State Variables, and Scales		
Aspect	**Relevance**	**Description**
Entities/Objects	Relevant	Network enclaves, services running on those enclaves, filters that prevent exploit arrival on the services, etc., are modeled as separate objects.
Spatial/Temporal Units and Scales	Relevant	Spacial scale does not matter, while temporal scale must be fine-grained enough to capture event arrivals.
Environment	Relevant	Network environment consists of multiple connected enclaves specified via an adjacency list or matrix.
Collectives	Irrelevant	Groups of objects are not modeled to have their own behaviors.

TABLE 3

The ODD protocol specification criteria for entities, state variables, and scales as they pertain to Network Porosity.

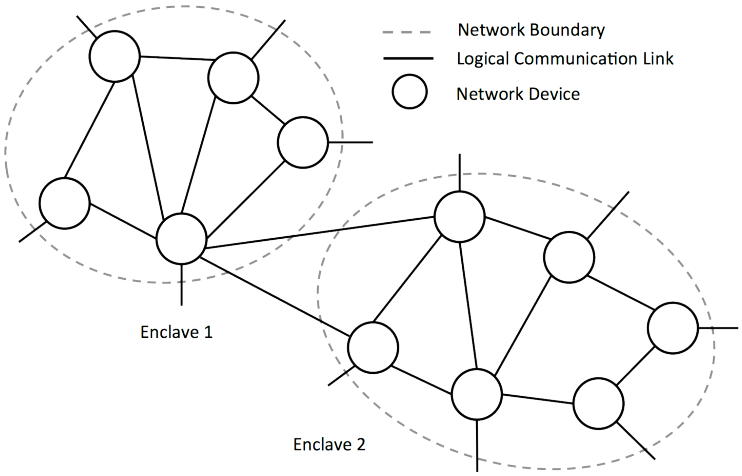

Figure 3. Two enclaves, that may belong to the same subnet and exchange communication, are separated from each other by their associated network boundaries. All traffic coming into or out of Enclave 1 must cross its boundary, and the same is true for Enclave 2.

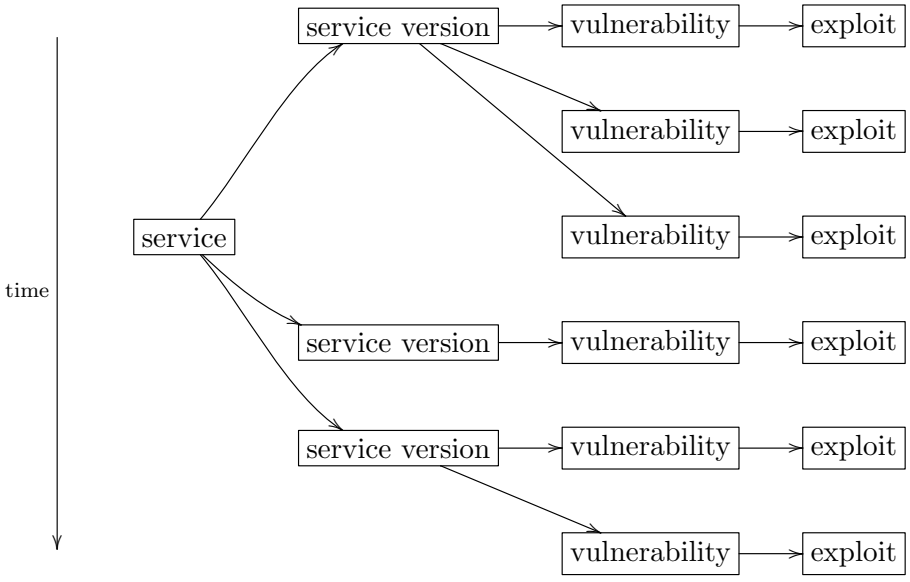

Figure 4. A service can have multiple versions and each version can have multiple vulnerabilities. However, each vulnerability has only one exploit.

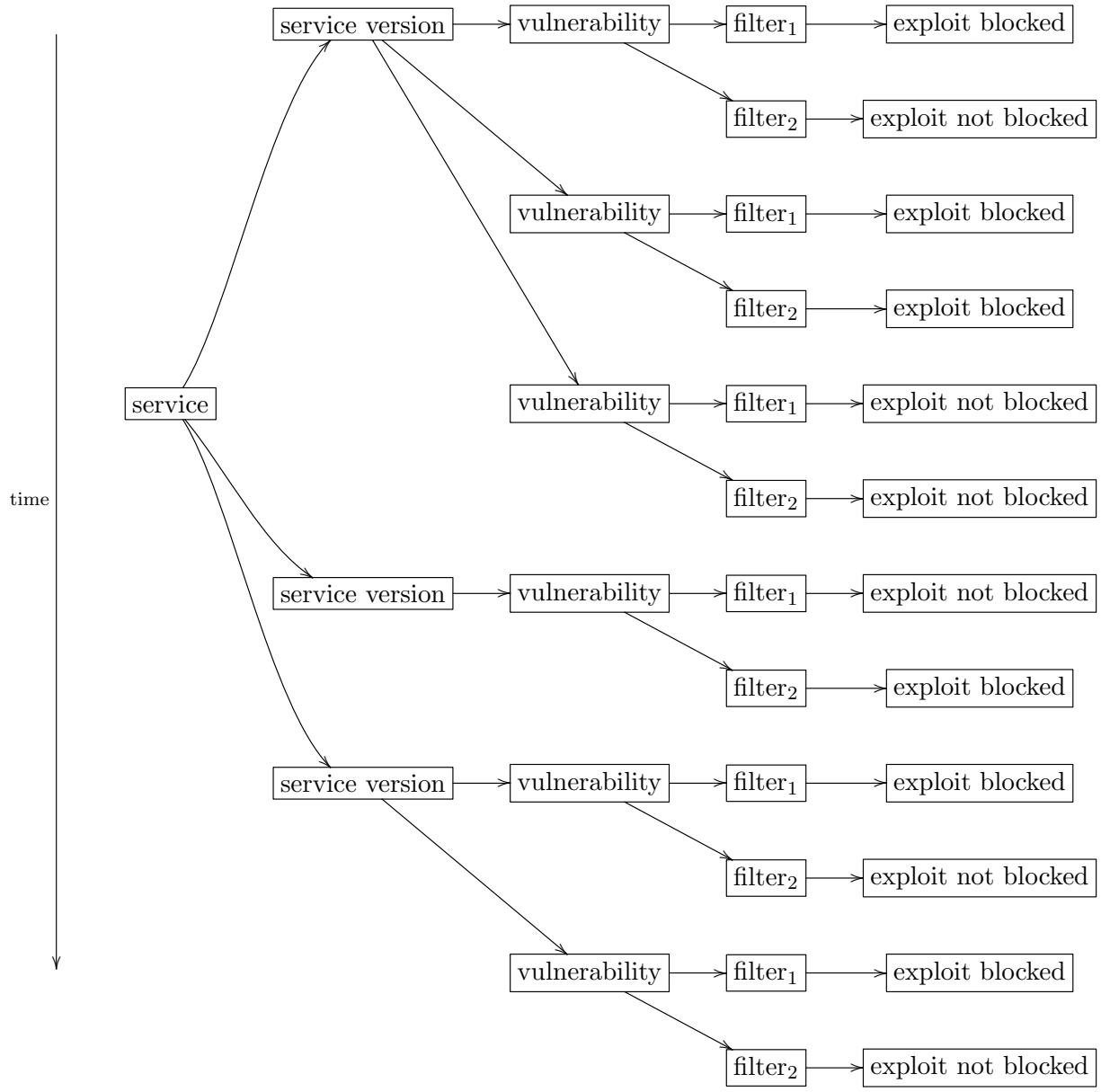

Figure 5. This figure is an extension to Figure 4 that includes the effect of two filters. Note how each filter can either block or not block an exploit for a particular vulnerability.

This page intentionally left blank.

4. SIMULATION EVENT DESCRIPTIONS

An important consideration for the implementation of any simulation model is that of computational efficiency. Two functionally identical models may have greatly varying performance (i.e. execution time) depending on how simulation events are processed. This section describes an approach to implementing the simulation that supports a form of *lazy evaluation* for simulation events. Lazy evaluation is a strategy that delays the evaluation of an expression until its value is needed and also avoids repeated evaluations of the same expression [3]. This approach diminishes the number of events that are processed and thus, can improve performance significantly. In practice, this approach is necessary to execute the simulation for non-trivial problem instances such that runs complete in a feasible amount time. We begin by motivating the approach with a simple example, including an analysis of execution time improvement, and continue to the general case.

4.1 ASSUMPTIONS AND TERMINOLOGY

We assume that the simulation will be implemented on an event based simulation engine that wherein we **schedule** events for execution at some future time. We will use the verb **set** to alter system state. We will use standard **if / else** constructions. We will use `typewriter` font to refer to objects, parameters and concrete terms within the simulation. We will use parenthetical composition to express parameters and references such as `event(parameter1, parameter2, reference1)`. We use $\Delta_{\texttt{name}}$ to refer to the mean of an exponential distribution, and sometimes as a shortcut to refer to a sample from that distribution.

4.2 EVENT EFFICIENCY

Consider a simple model where two independent Poisson processes flip the state of a system between red and blue. This system can be simulated with or without lazy evaluation.

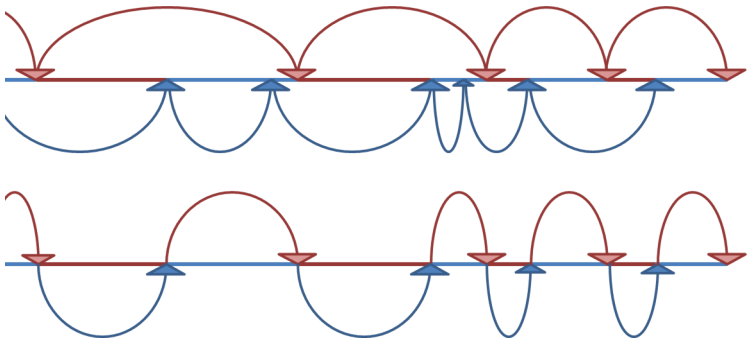

Figure 6. Two Implementations of Two State System

Algorithm 1 Non-Lazy Red-Blue Event Scheduling

```
function EVENT(color, Δ_color)
   system-color ← color
   schedule event(color, Δ_color) at Δ_color
end function

function INITIALIZE
   s ← single sample from U(0,1)
   if Δ_blue/(Δ_red+Δ_blue) > s then
      system-color ← red
   else
      system-color ← blue
   end if
   schedule event(red, Δ_red) at Δ_red
   schedule event(blue, Δ_blue) at Δ_blue
end function
```

4.2.1 Non-Lazy Evaluation

In the non-lazy setting we have two recurrent processes that are scheduled at a time sampled from an exponential distribution with mean $\Delta_{\texttt{color}}$. Algorithm 1 defines those processes where $\frac{\Delta_{\text{blue}}}{\Delta_{\text{red}}+\Delta_{\text{blue}}}$ gives the expected fraction of time a system will be red and $U(0,1)$ is a uniform distribution from 0 to 1. The resultant event sequence is depicted in the upper half of Figure 6. Note that there is a racing condition involving two independent processes modifying variable system-color that could in principle be addressed by sampling according to steady state probabilities. In practice, however, we do not expect such a condition to occur often enough to make a non-negligible impact.

4.2.2 Lazy Evaluation

In the lazy evaluation setting we have a single alternating recurrent process that is scheduled at a time sampled from an exponential distribution with mean $\Delta_{\texttt{color}}$ or $\Delta_{\texttt{next-color}}$. In the general case this would require generation of new distributions. In the Poisson setting it does not due to the memory-less property of exponential distribution. Algorithm 2 defines the alternating process where $\frac{\Delta_{\text{blue}}}{\Delta_{\text{red}}+\Delta_{\text{blue}}}$ gives the expected fraction of time a system will be red and $U(0,1)$ is a uniform distribution from 0 to 1. The resultant event sequence is depicted in the lower half of Figure 6. Note that the issue with a racing condition present in Algorithm 1 does not arise in Algorithm 2 because the latter uses only a single process.

The non-lazy (Algorithm 1) and lazy (Algorithm 2) implementations result in identical red-blue state distributions but the lazy evaluation implementation will have fewer events. If Δ_{red} and Δ_{blue} are means of the two processes and w is the duration of the experiment, then non-lazy implementation should expect

$$\frac{w}{\Delta_{\text{red}}} + \frac{w}{\Delta_{\text{blue}}}$$

Algorithm 2 Lazy Red-Blue Event Scheduling

```
function EVENT(color, Δ_color, next-color, Δ_next−color)
   system-color ← color
   schedule event(next-color, Δ_next−color, color, Δ_color) at Δ_next−color
end function

function INITIALIZE
   s ← single sample from U(0, 1)
   if Δ_blue / (Δ_red+Δ_blue) > s then
      schedule event(red, Δ_red, blue, Δ_blue) at Δ_red
   else
      schedule event(blue, Δ_blue, red, Δ_red) at Δ_blue
   end if
end function
```

events whereas the lazy implementation should expect

$$\frac{2 \cdot w}{\Delta_{\text{red}} + \Delta_{\text{blue}}}$$

events so that the ratio of non-lazy to lazy events is

$$\frac{1}{2} \cdot \left(\frac{\Delta_{\text{red}} + \Delta_{\text{blue}}}{\Delta_{\text{red}}} + \frac{\Delta_{\text{red}} + \Delta_{\text{blue}}}{\Delta_{\text{blue}}} \right)$$

whose value is large when the two rates are significantly different. For example if Δ_{red} is 1 week and Δ_{blue} is one second, the lazy optimization results in a speedup of 302,401 times.

We will utilize this laziness

4.3 NETWORK POROSITY EVENTS

The simulation can be coded in a similarly lazy manner by triggering events from one another rather than fully asynchronously. This provides significant performance gains due to the significantly different time scales with which various events take place (e.g. compromise time of milliseconds and enclave clean time of years).

Figure 7 depicts the compromise propagation model of network porosity. A compromise will flow (spread from one enclave to another) only under the following four conditions:

- there is an attacker on a compromised enclave with a connection to an uncompromised enclave via a service,
- the service has a current vulnerability,
- the attacker has an exploit for that vulnerability, and

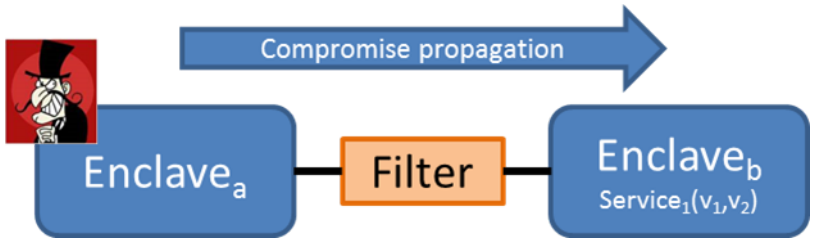

Figure 7. Compromise Propagation

Algorithm 3 Vulnerability and Patch Processes

```
function SERVICE-VULNERABILITY-EVENT(service, vulnerability)
   vulnerability-list ← vulnerability-list ∪ vulnerability
   schedule exploit-development-event(service, vulnerability) at Δdev
   schedule service-vulnerability-event(service, vulnerability) at Δvuln
   if len(vulnerability-list) == 1 then
      schedule service-patch-event(service) at Δpatch
   end if
end function

function SERVICE-PATCH-EVENT(service)
   increment version
   empty vulnerability-list
end function
```

- the filter does not block that exploit.

As such there are going to be three distinct cases where we have to consider the compromise propagation:

1. the source enclave is newly compromised,
2. the destination enclave is newly cleaned, or
3. a new exploit for a service is developed.

In each case, we can speculatively schedule compromise propagation events *subject to the four conditions* when such a speculative compromise propagation event from the source to the destination via the service is not already scheduled[1]. When the speculative compromise event is executed, it tests that the attacker has a current exploit for the service that is passed by the filter and that the source enclave is still compromised.

[1] New vulnerability or exploit arrival does not change the frequency at which the source enclave contacts the destination enclave.

The two processes in Algorithm 3 define a simplistic model of the arrival of vulnerabilities and patch releases where each `vulnerability` has an associated exploit development occurrence interval ($\Delta_{\texttt{dev}}$) and each `service` has an associated `version`, `vulnerability-list`, occurrence interval of vulnerability discovery ($\Delta_{\texttt{vuln}}$), and occurrence interval of patching a particular `vulnerability` ($\Delta_{\texttt{patch}}$). In variations of the metric we could create more sophisticated models that, handles vulnerabilities spanning multiple versions of a package, releases without any vulnerabilities, or creation of multiple exploits for a single vulnerability but for the purposes of this metric we have chosen simplicity.

The processes in Algorithm 4 define a simplistic model of the exploit development and propagation processes where each `service` has an associated `vulnerability-list`, and each (`source`, `destination`, `service`) triplet is associated with an occurrence interval of compromise propagation ($\Delta_{\texttt{propagate}}$).[2] Note that if multiple services are vulnerable then multiple `propagate-compromise-event(...)` processes may be scheduled between a single `source` and `destination`, but multiple vulnerabilities in a `service` can result in only a single scheduled `propagate-compromise-event(...)`. Also, the condition to check for exploit filtration was placed in `propagatecompromise-event(...)` to ease the development of possible filter signature models in later versions of the metric.

The process in Algorithm 5 defines a simplistic model of what happens when an enclave is cleaned where each `service` has an associated `vulnerability-list`, and each (`source`, `destination`, `service`) triplet is associated with an occurrence interval of compromise propagation ($\Delta_{\texttt{propagate}}$). Note that if multiple services are vulnerable then multiple `propagate-compromise-event(...)` processes may be scheduled between a single `source` and `destination`, but multiple vulnerabilities in a `service` can result in only a single scheduled `propagatecompromise-event(...)`.

[2] We assume that any racing conditions between the independent processes will not arise often enough in practice to have a non-negligible impact.

Algorithm 4 Exploit Development and Propagation Processes

```
function EXPLOIT-DEVELOPMENT-EVENT(service, vulnerability)
   if vulnerability in vulnerability-list then          ▷ service still vulnerable
      for all source in compromised-enclaves do
         for all destination in uncompromised-reachable-enclaves do   ▷ reachable
via service
            if propogate-compromise-event(source, destination, service) not in
scheduled then
               schedule propogate-compromise-event(source, destination,
service) at Δ_propagate
            end if
         end for
      end for
   end if
end function

function PROPAGATE-COMPROMISE-EVENT(source, destination, service)
   if attacker-has-exploit and exploit-not-filtered and (source in
compromised-enclaves) then
      add destination to compromised-enclaves
      attack-from-newly-compromised-enclave(destination)
   end if
end function

function ATTACK-FROM-NEWLY-COMPROMISED-ENCLAVE(source)
   for all destination in uncompromised-reachable-enclaves do
      for all service in destination do
         if vulnerability-list not empty then                    ▷ of service
            if (source, destination, service) not in compromise-event-schedule
then
               schedule propagate-compromise-event(source, destination,
service) at Δ_propagate
            end if
         end if
      end for
   end for
end function
```

Algorithm 5 Post-Clean Compromise Process

```
function ENCLAVE-CLEANED-EVENT(destination)                ▷ input destination is clean
   for all source in compromised-and-can-reach-destination do
      for all service in destination do
         if vulnerability-list not empty then                                ▷ of service
            if (source, destination, service) not in compromise-event-schedule
then
               schedule propagate-compromise-event(source, destination,
service) at Δ_propagate
            end if
         end if
      end for
   end for
end function
```

This page intentionally left blank.

5. PUBLICATIONS WITH FURTHER DETAILS

1. Suggestions on designing a secure network using enclaves, filtering, and proxies are available in the NSA. Manageable Network Plan [2].

2. Suggestions on managing unauthorized devices are also available in the description of Critical Control 19 in SANS CSIS-20 Critical Security Controls Version 4.1 [8].

3. The technical report [7] describing the first four metrics is available as well as a more recent treatment of the matter [6].

This page intentionally left blank.

REFERENCES

[1] Volker Grimm et al. The ODD protocol: review and first update. *Ecological Modeling*, 221:2760–2768, 2010.

[2] The Mitigations Group. Manageable Network Plan. Special publication, National Security Agency, 2012.

[3] Paul Hudak. Conception, evolution, and application of functional programming languages. *ACM Computing Surveys*, 21(3):383–385, September 1989.

[4] Kyle Ingols, Matthew Chu, Richard Lippmann, and Stephen Boyer. Modeling Modern Network Attacks and Countermeasures Using Attack Graphs. In *ACSAC Proceedings*, 2009.

[5] R. P. Lippmann and K. W. Ingols. An annotated review of past papers on attack graphs, 2005.

[6] Richard Lippmann, James Riordan, Sebastian Neumayer, and Neal Wagner. Introduction and Overview to Maturity Metric Descriptions. Technical report, MIT Lincoln Laboratory, 2014.

[7] R.P. Lippmann, J.F. Riordan, T.H. Yu, and K.K. Watson. Continuous Security Metrics for Prevalent Network Threats: Introduction and First Four Metrics. Technical report, MIT Lincoln Laboratory, May 2012.

[8] SANS Institute. Critical Security Controls Version 4.1. Technical report, SANS Institute, 2012.